CS457/657 Database Management Systems

Programming Assignment 1: Metadata Management

Overview

In this assignment, you will write a program that allows a database user to manage the metadata of their relational data. By metadata, we mean the database's own information (e.g., database's name, creation time, owner) as well as the properties of the tables (e.g., table's names, attributes, constraints).

System Design

- You are free (in fact, encouraged) to come up with your own design
    - For instance, Sqlite3 uses one single file for each "database."
- Here is one possible design:
    - One Linux directory -> a database
    - One regular file -> a table

Implementation

- The program should not use an external *database* library or tool.
- Any programming language is acceptable, e.g., Python, Java, C/C++, Go
    - Please pick one that you are most comfortable/proficient with
    - If you want to choose a language not mentioned above, please contact the TA *before* you start coding
- Functionalities:
    - Database creation, deletion
    - Table creation, deletion, update, and query

Interface

- A similar but simpler interface than Sqlite3
- Examples (on a Linux terminal):
    - # ./<your_program> <enter>
      # CREATE DATABASE db_name <enter>;
        - The shell should prompt whether the command is successful or failed
            - If failed, don't crash but gracefully prompt why
        - Then when you check your file system, it might look like this:
            - ~/your_home/cs457/pa1/db_name
    - # ./<your_program> <enter>
      # USE db_1; CREATE TABLE test_tbl (a1 int, a2 char(9));
        - If successful, then your file system might look like this:
            - ~/your_home/cs457/pa1/db1/test_tbl

Testing

- We will test your program on Ubuntu (version 14 or above)
- If your program cannot compile on our testbed, we may ask you to demo your program
    - Hint: try not to use many exotic libraries
- A full test script will be provided
    - # ~/cs457/pa1/<your_program> < PA1_test.sql (expect the standard input)
        - Alternatively, you can use a file name as an argument to your program.
        - You will NOT lose points by only supporting a filename-argument interface, but keep in mind that the *standard input* interface would be more desirable for your users (e.g., our TA, Abdullah).
    - # <expected output, hopefully…>
    - You don't need to parse the comment lines (i.e., starting with "- -")
    - We will not to test your programs with any other scripts/commends
        - However, it's always good to consider more edge cases
        - Try not to hardcode your parser:
            - You want to parse them into a series of (dynamic) words

Grading (20 points total)

- This is an individual assignment.
- Design document that clarifies the followings: (5 points)
    - How your program organizes multiple databases
    - How your program manages multiple tables
    - At a very high level, how you implement those required functionalities
- Source code (15 points)
    - Coding style and clarity, 5 points
        - Appropriate parenthesis locations, indention, etc.
        - Always write comments at the beginning of any files
            - Author, date, history, etc.
        - Always write comments at the beginning of any non-trivial class/function
            - What this class/function does, high-level algorithm if needed
        - Write in-line comments for non-trivial blocks of code
    - Functionality, 10 points
        - Refer to the test script for detailed breakdowns

Submission

- WebCampus
- Compress all your source code and report into one package in this format:
    - <your_netid>_pa1
- Late penalty: 10% per day